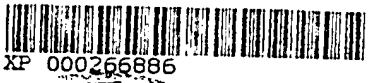
Future Generation Computer Systems 7 (1991/92) 329–337
North-Holland



Transaction models for federative distributed database systems

Wolfgang Johannsen

IBM Germany, European Networking Center, Tiergartenstrasse 8, W-6900 Heidelberg, Germany

Abstract

Johannsen, W., Transaction models for federative distributed database systems, Future Generation Computer Systems 7 (1991/92) 329–337.

Federative Distributed Database System (FDDBS) are composed out of independent nodes in a computer network. Each server node comprises a Database Management System (DBMS) offering services of different quality to the outside world, i.e. remote users at client systems. In addition local services are offered to local users. Server nodes are independent from other notes. This reflects the characteristic of an dynamically growing and shrinking federation of databases in open systems. Typically user applications are distributed and should be supported by management functions that guarantees certain service quality. The most important are atomicity and isolation. Units of works that provides such characteristics are distributed transactions.

In this paper we discuss the advantages and disadvantages of two transaction models with respect to FDDBSs. Beside traditional distributed (global) transactions we consider a model based on the Saga approach. We show how both can be integrated in one environment. The approach taken increases node autonomy and allows for a higher degree of parallelism without loss of the advantages of traditional distributed transactions.

Keywords. Distributed transactions; node autonomy; federative systems; serializability; sagas.

1. Introduction and overview

One of the significant characteristics of future open systems will be the ability to support distributed database applications. The databases to be accessed will be managed by heterogeneous Database Management Systems (DBMS) running in open systems environments. Each node in such a network requires a great amount of node autonomy with respect to availability, security, defining access rights and controlling of external (remotely) requested applications [1]. We will concentrate on execution autonomy, that is the freedom of a node to run an application on a node without becoming dependent on other nodes. Refer to [2] for an architecture based on ISO/OSI protocols that overcomes network and database heterogeneity.

Data maintained by different Database Management Systems (DBMS) is independent from each other. Consequently no consistency constraints beside such required by a running application exists across network node boundaries.

One more important characteristic besides heterogeneity and autonomy will be the absence of a global database schema that would allow for defining integrity constraints or maintaining referential integrity across nodes. Interconnected database systems with such characteristics form a Federative Distributed Database System (FDDBS). The term *Federation* refers to the relatively high degree of autonomy and low degree of interdependencies between nodes.

Distributed applications in a FDDBS will rely on distributed transactions. Because of the potential conflict between autonomy requirements and the realization of traditional distributed transactions, new transaction execution strategies and transaction models have to be considered.

The common approach to implement dis-

8261 Future Generation Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

tributed transactions is based on a combination of Two-Phase-Locking (2PL) [3] and Two-Phase Commit (2PC) [4]. The first provides for local concurrency control and the latter for coordinated termination of subtransactions. It is assumed here that a so called Global Transaction (GT) consists of subtransactions (limited to one per node). The combination of 2PL/2PC provides global serializability [5].

Beside this traditional approach, new transaction models and execution strategies are being designed. In [6], [7] the synchronization goal is relaxed by introducing *quasi serializable* histories of operations. This is achieved when local transactions of a node are not considered as candidates for synchronization with subtransactions of distributed Global Transactions.

Several approaches have been introduced to relax the 'isolation' between distributed transactions. The best known is the Saga approach [8]. Subtransactions of Sagas are executed independently. The results of subtransactions are made visible before the last subtransaction of a certain Saga is executed successfully. Thus node autonomy is increased since local data can be accessed independently from the state of other nodes. In case of failures compensating transactions are used for recovery. Related approaches are S-Transactions [9] and Migrating Transactions [10].

In this paper we introduce the concept of Global Transaction Procedures (GTP). GTPs are global units providing for *semantic atomicity* [11]. Subtransactions of different GTPs are executed completely isolated from each other whereas at the global level the isolation is reduced. GTPs are integrated with GTs in order to overcome major drawbacks of Sagas. In particular it is the restriction to 'simple' distributed applications that is overcome by the integration of GTPs with GTs in a single environment. As a result the integrated approach allows for execution of complex transactions based on traditional GTs. The simpler semantics (ordering, reservations, preparing schedules etc.) are based on GTPs.

The section following this overview gives an outline of our architecture model of an open FDDBS. Then Global Transactions and serializability requirements are discussed. Global Transaction Procedures are discussed next. Our approach of integrating these transaction models is presented in the final part of the paper.

2. The architectural model of an open federative database system

According to the client/server model database *server systems* in our environment offer services accessible by *client systems*. The client systems (workstations and mainframes) may or may not maintain their own database systems. Database server systems are administered by human administrators responsible for an *administrative area* that may span several server systems. For example in *Fig. 1* an administrative area could include all servers (S) within a certain Local Area Network (e.g. NW 3). Such an area consisting of several database systems may form a distributed database system of its own. In the latter case we assume that a single service interface for remote users and a single global conceptual schema that help to maintain integrity constraints is provided for each administrative area. *Databases in open systems* are accessed by distributed applications. The number of databases accessible by remote users in such a configuration changes dynamically. Databases can be made accessible and removed at any time. No global administration is responsible for maintaining the whole system. Heterogeneity (*Fig. 2*) is overcome by standardized software components in open systems. Communication protocols standardized by ISO according to the reference model for

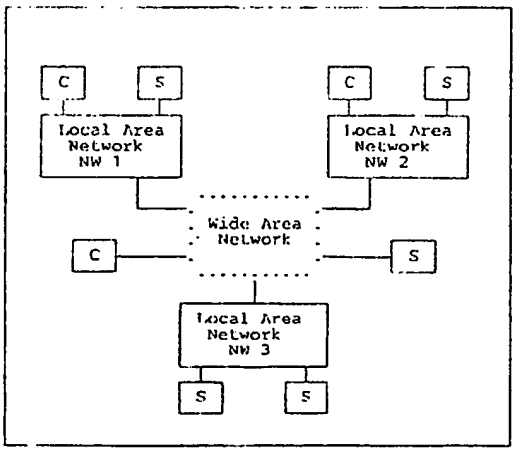


Fig. 1. *Federative Distributed Database System*: The figure shows an example configuration of a FDDBS consisting of client (C) user nodes and server (S) database system nodes.

8261 uture Generation Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

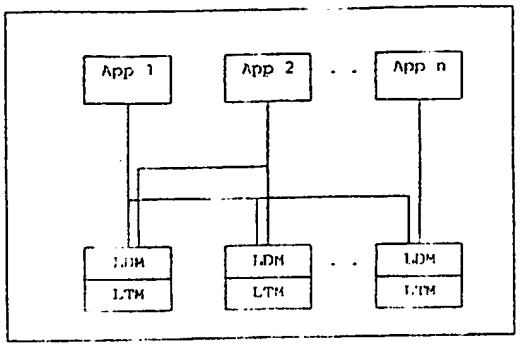


Fig. 2. *Open Systems*: Distributed applications are executed without support of global centralized components.

Open System Interconnection (OSI) [12] like Remote Database Access (RDA) [13] and Transaction Processing (TP) [14] play a key role here [2]. The a priori decentralized character of open system environments prohibits the application of centralized control functions as GM components. The more natural solution is to decentralize control software as much as possible.

Obviously data stored in different databases in open systems is independent from each other.

Server nodes running a DBMS comprising components for local transaction management (LTM) and local data management (LDM). In addition, an interface for global transaction support (i.e. a Two-Phase-Commit Protocol interface, 2PC) has to be provided. This interface is connected to the local communication subsystems (*Fig. 3*).

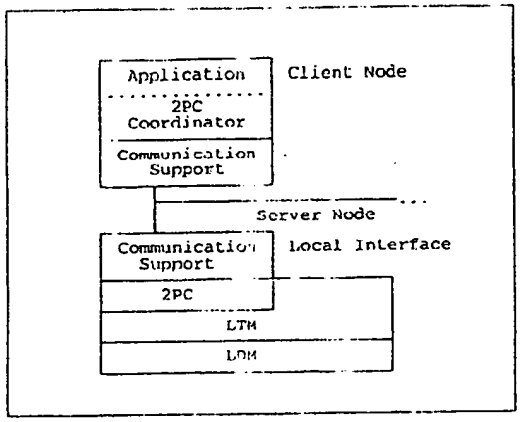


Fig. 3. *Node Architecture*: Each Node provides for a Two-Phase-Commit (2PC) protocol interface for subtransactions of Global Transactions. The coordination is in the responsibility of the client nodes.

Servers involved in a distributed application do not cooperate directly. However, we do not restrict a server from acting as a client with respect to other servers thereby forming a tree like transaction structure [14].

Client nodes act as *initiators* and *coordinators*, responsible for one or more applications accessing remote server databases. In general a Global Transaction is initiated and controlled by a client component. To fulfill this task the client might ask for more services than are provided by databases to be accessed. In particular dictionary services are needed to allow a certain degree of location transparency.

Client *applications* are *global*. This is by means of accessing more than one remote system at a time.

Based on the assumptions above, we list the main characteristics of our architectural model of an open FDDBS:

1. *Decentralized control functions*:
   - Each (human) database administrator is responsible for only a subsection of all database systems accessible by a global (distributed) application.
   - Transaction management is done by the local database management systems in cooperation with coordinators of global applications. Coordination functions are needed for termination of Global Transactions.
2. *Independence of coordinators*: Coordinators do not exchange messages.
3. *Local DBMSs support of transaction termination*: An interface for supporting the Two Phase Commit protocol is to be provided at each local DBMS. In case different transaction models are applied additional termination support is required. Termination support is the only extension to existing database management systems that has to be provided in order to integrate databases into a federation.
4. *Treating subtransactions of Global Transactions*: Subtransactions of Global Transactions are treated the same way as local transactions.

Both types of transactions are managed by the same local concurrency control mechanism. Therefore from a local point of view no

8261 Future Generatio. s Co.pute Systems
7(1992)April, Nos.2/3, Amsterdam, NL

differences between local transactions and subtransactions exists.

5. *Execution Autonomy*: Local database systems are not requested to operate all the time.
6. *Independence of data*: Data maintained by local databases is considered to be independent from each other.

We are not considering replicated data in our model. This is due to the fact that we see a open system as a dynamically growing and shrinking collection of independent database systems.

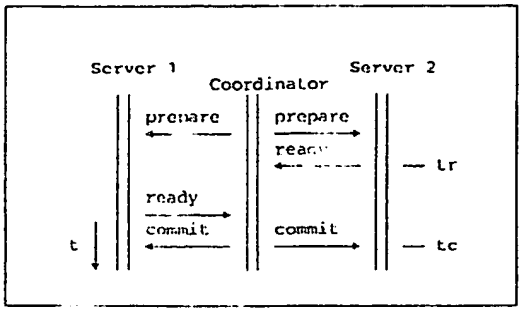


Fig. 4. *Autonomy*: Between tr and tc the server has to wait for the decision of the coordinator.

3. Global transactions

A widely accepted method of realizing Global Transactions is the combination of Two-Phase-Locking (2PL) and Two-Phase-Commit (2PC) Protocols. 2PL protocols provide for local serializable schedules. In our model local transactions and subtransactions of global transactions are executed concurrently at the same node. Thus they are treated by the same concurrency control and recovery system at each local database.

Global consistency is preserved by coordinated termination of global transactions. Termination of Global Transactions is supported by the 2PC protocol. The 2PC/2PL combination guarantees that locks held by the local concurrency control components on behalf of a subtransaction of a Global Transaction are not released before the coordinator explicitly orders to do so. The coordinator observes the state of all subtransactions and ensures that each one can be terminated successfully (commit) only if all other subtransactions are also ready to commit.

It is well known that the 2PC/2PL combination produce global serializable schedules. It is obvious that at each node the same relative order of subtransactions is produced. Otherwise a cycle in the dependency graph that could lead to a distributed deadlock would occur [15]. This is true even for the combination of 2PC and local optimistic concurrency control [5].

Nevertheless, the use of 2PC protocol has some drawbacks which are especially harmful in open systems. Local DBMSs are made to wait for each other for locally unknown time periods because of the 2PC voting procedure. In addition distributed deadlock may occur.

Autonomy: As explained before the 2PC protocol is used to coordinate termination of Global Transactions. A Global Transaction can only be committed if all subtransactions on all nodes have sent a *ready* message (*Fig. 4*). Otherwise all subtransactions have to be aborted. This implies that servers wait for each other. The latest subtransaction entering the ready state determines the time when the decision to commit can be made by the coordinator. Until that time all other servers have to wait. The servers are not allowed in this state to release locks or cancel the subtransactions based on their own decision. In other words they are loosing the autonomy of executing applications on data locked by waiting subtransactions of Global Transactions.

4. Serializability requirements in open systems

Global serializability for most applications is a sufficient correctness criterion. It guarantees that the result of a series of Global Transactions is the same as if they would be executed in strong (arbitrary) serial order.

Strict isolation of transactions is the main prerequisite to gain this property. Isolation of Global Transactions prevents the reading of any uncommitted results of a certain transaction by other transactions.

It has often been argued that global serializability is too strong a criterion for correctness in federative distributed databases [6,8,10,16]. The main reason for this is the reduced execution autonomy resulting from using the 2PC protocol during transaction termination. The approaches

8261 Future Generations Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

suggested so far to overcome this are based mostly on *semantic atomicity* [11].

Although the proposed solutions increase the node autonomy they are restricted to a small class of applications of rather simple semantics (ordering, reservations, preparing schedules etc.) [16]. This is because the execution of Global Transactions is controlled by execution plans specified by a dedicated language or other description techniques. The execution plans include instructions for subtransaction execution order, descriptions of allowed interleaving transactions and rules for the case of a failure.

In our approach we also claim that in cases where the databases are independent from each other, global serializability is not needed for all applications. Here the most important property of global transactions is atomicity instead of isolation. Although reduced isolation implies loss of global serializability, local serializability is still required for subtransactions and local transactions.

Example. A machine producing company assembles products of parts produced by subcontracters. Each company producing parts runs a database system that allows for remote access. In order to keep the production plant supplied optimally with required parts an automatic ordering system is used. The system orders parts by using remote databases controlled by the producing companies. The restriction is: an application is considered to be successful only if all parts needed for a machine to be assembled can be ordered. This is to reduce storage requirements for the company as much as possible. As a consequence the application has to be executed as an atomic unit of work. In case a requested part is not available an alternative company (database) is used. In case no alternatives are at hand the orders already done are canceled.

When analyzing this example (characteristics are adopted from [17]) it is obvious that the application requirements could be met by an atomic Global Transaction. The transaction properties make sure that either each order issued by the company is successfully executed or none is valid. However, since the accessed databases are completely independent from each other no need to isolate concurrent global applications of this type at the global level exists. In other words, the Global Transactions are over qualified for the

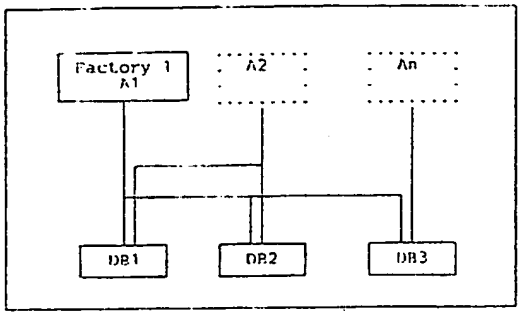


Fig. 5. *Factory application*: An atomic application accesses several remote databases.

example application. More precisely, the isolation between Global Transaction can be reduced.

Reduced isolation at the global level leads immediately to well-known drawbacks. Let us assume that an application A1 (*Fig. 5*) starts at server DB1 and orders the last available part there. At the same time application A2 may start at server DB2 and orders the last available part. In case A1 and A2 have to make orders at nodes DB2 and DB1 respectively both are unsuccessful in the end. Compensating would be necessary whereas Global Transactions would have been aborted (because of a deadlock) if they would have been executed the same way.

However in the latter variant of our example, database consistency can not be affected provided all local applications are executed as local transactions. Thus we consider the above mentioned drawback as comparatively small since node autonomy is improved and distributed deadlocks cannot occur by following this execution scheme.

Isolation can be reduced only at the global but not at the local level. Locally each issued order (and the cancellation of orders) is a local transaction running under control of the local concurrency mechanism in parallel to other subtransactions of Global Transactions or local transactions.

The example shows that the need to execute this application in full isolation from others does not exist. We need to make sure that at each state of execution of applications the *all or nothing* property (atomicity) can be guaranteed. This is to be done by calling compensating applications and, if at hand, alternative applications. Isolation would introduce some drawbacks:

8261 Future Generations Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

1. other customers would have to wait for uncertain amount of time before being allowed to access the data requested, and
2. communication links would have to be held for each node.

However, not all types of applications should be executed with reduced isolation. For example, interleaved applications that compute the worth of all parts available need to be isolated from other applications. This is in order to guarantee that a stable global database state can be seen. Only this allows the making of a correct account that does not include orders of partially executed applications. In case this type of application would run with reduced isolation the client would get wrong results but still no consistency violation would occur since local databases would not be changed. In many cases (e.g. compute a rough estimate of the stock situation) a user might be statisfied with a result computed this way.

5. Global Transaction Procedures (GTPs)

We have shown that global applications in FDDBs need not always be executed under the restriction of guaranteeing global serializability. As the former example shows, certain applications can be executed as atomic units without being isolated from each other at the global level. In case execution autonomy of nodes is an important design goal, reduced isolation at the global level would relax the drawbacks introduced by Global Transactions.

A new model of a Global Transaction, the Global Transaction Procedure (GTP) serves this purpose and will be introduced in the following.

GTPs are based on *Sagas*. In [8] Sagas are introduced as a collection of intended subtransactions that can be interleaved in any way with other transactions. To ensure atomicity the authors require that *compensating subtransactions* (cST_i) are used (*Fig. 6*).

Once compensating transactions, cST_1, cST_2, ..., cST_{n-1} are defined for Saga ST_1, ST_2, ..., ST_n, then the system can make the following guarantee: either the sequence ST_1, ST_2, ..., ST_n or the sequence ST_1, ST_2, ..., $ST_j cST_j$, ..., cST_2, cST_1 for some $1 \leq j < n$ will be executed [8].

The results of the subtransactions are made immediately visible for other Sagas when a subtransaction finishes execution. Since no 2PC protocol is used, the isolation between Sagas is restricted to isolation between subtransactions on local systems.

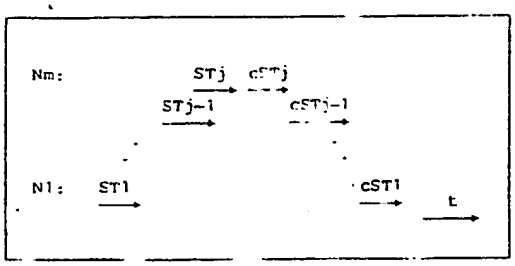


Fig. 6. *Compensating Subtransactions*: STj produces an undesired result at node Nm. The subtransactions executed so far are compensated.

In [17] Sagas are applied to federative distributed database management systems. In our approach we extend the proposed model by:

1. integrate it with the traditional model of distributed transactions (Global Transactions) and
2. integrate both into one database programming language DBPL [18,19].

In the following we will concentrate on the synchronization aspects by integrating both transaction models into one system environment. Refer to [19] for the language aspects.

In addition Global Transaction Procedures comprises alternative subtransactions. All subtransactions are executed in a way that allows for order independent compensation of subtransactions.

We will explain the mentioned properties of GTPs more detailed in the next paragraph.

5.1. Alternative subtransaction

In case a subtransaction is not available e.g. because of failures or inaccessible resources (nodes, databases etc.) we allow for the execution of alternative subtransactions [8].

This reflects the expected services of a FDDBS. In an open system services may be replicated on different server nodes. In case a subtransaction aborts it is quite natural to look for a similar service (e.g. a flight is booked out and an alternative airline is taken). Alternative subtransactions remove the necessity of running compen-

8261 Future Generations Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

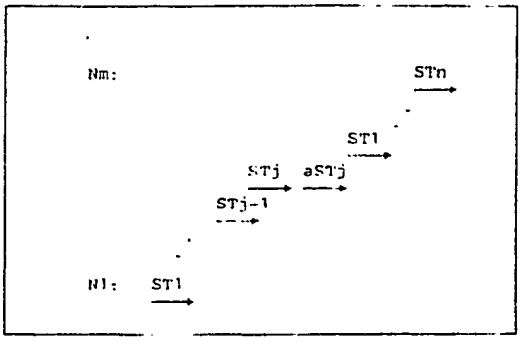


Fig. 7. *Alternative Subtransactions*: STj produces an undesired result at node Nj. Instead of compensating already produced results the alternative subtransaction aSTj is executed automatically.

sating subtransactions for those subsets of subtransactions that are already completed successfully (*Fig. 7*). Thus on average, a smaller set of compensating subtransactions have to be run, resulting in an increased throughput of intended subtransactions. The model requires that for each alternative subtransaction a compensating subtransaction be provided. Thus an alternative subtransaction can dealt with the same way as an intended subtransaction.

5.2. Order independence

Subtransactions belonging to GTPs should be treated as ordinary local transactions. Thus the isolation between GTPs is reduced. They commit immediately after finishing computing and make their results visible to other GTPs. However the execution of a GTP should obey certain consistency requirements.

As already mentioned, in our model GTPs are built out of subtransactions exported by server databases forming the FDDBS. Exportation of subtransactions refers to the decision of the database (human) administrator of a certain *administrative area* to make a set of prepared subtransactions accessible for remote users. The administration of server databases offer only such subtransactions to be imported into a GTP that fulfill basically the following property:

Let $GTP_1 = \{ST_{1a}, cST_{1a}, ST_{1b}, \ldots\}$ and $GTP_2 = \{ST_{2a}, cST_{2a}, ST_{2b}, \ldots\}$ be GTPs consisting of subtransactions ST_{ij} to be executed on nodes N_j. Then the administration of a node N_j allows only those subtransactions to be exported into GTPs that are order independent.

$$ST_{1j} \rightarrow ST_{2j} \rightarrow cST_{2j}$$
$$\Leftrightarrow$$
$$ST_{2j} \rightarrow cST_{2j} \rightarrow ST_{1j}$$
$$\Leftrightarrow$$
$$ST_{2j} \rightarrow ST_{1j} \rightarrow cST_{2j}$$
$$\Leftrightarrow$$
$$ST_{1j}$$

The above implies that the final outcome is independent of the order in which the intended subtransactions and compensating subtransactions are executed. The result left within the database has to be the same as if the intended subtransaction ST_{1j} were executed alone. Note that it is not allowed for a compensating subtransaction to be executed in precedence to its intended counterpart.

However a drawback of the usage of GTPs should be mentioned here: The (human) database administrator responsible for the semantics of exported subtransactions has to make sure that the order independence of all exported subtransactions is valid. Therefore order independence usually is reachable only for certain 'simple' semantics of subtransactions and thus only a subset of database applications can be executed under this assumption.

Order independence is of local matter to the server nodes and thus only subtransactions of a certain server are considered by proving this criterion.

We have to take into account that order independence can not be proved in most cases. This is because most applications do not allow order independent executions in arbitrary order. In addition for complex applications it would often mean an unacceptable amount of work to prove this condition. Instead the administrator has to run tests of interleaving conflicting subtransactions and compensating subtransactions.

The advantage of order independence is that an administrator controls the semantics executed by GTPs on his node. By exporting subtransaction into Global Transaction Procedures he is

8261 Future Generations Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

provided with a higher degree of node autonomy without loosing the control of local consistency.

5.3. Synchronizing GTs and GTPs

In this paragraph we give an outline how the integration of both transaction models is done. Our approach to increasing node autonomy restricts applications to global semantic atomicity where possible. Global Transactions are only used where necessary [19].

The main advantage of this approach is to provide the user with the freedom to make a choice of an appropriate execution support. Of course this is restricted to a certain extent. The main restriction concerns the semantics of applications executed with semantic atomicity. Since no global serializability is provided the applications have to be selected carefully. The second restriction concerns the right to define the semantics of that class of applications by programming the subtransactions to be exported. We will assume that only the provider of a database service is allowed to define the semantic of (sub)transactions exported to GTPs.

The main goal for the synchronization of GT and GTP in one environment is to prevent GT from reading data written by subtransactions of GTPs that are active concurrently. If this would be allowed the data read is in danger of being the subject of a later compensating subtransaction. This data therefore can be seen as 'dirty data' since subtransactions of a GT are not executed to be order independent. In case GTs would be allowed to read this data, the problem of cascading rollbacks would arise.

We enforce the synchronization of GT and GTP by checking whether or not GTs or GTPs are active on data to be accessed. This requires that GTPs announce themselves before beginning a subtransaction at a certain node. The active state is relevant to the data scope being accessed. Thus preclaiming (for subtransactions of GTPs only) is used to define the scope to be accessed in advance. A GTP is deactivated by a message after the last subtransaction of this GTP is executed successfully. This procedure provides for a mutual exclusion of GT and GTP when conflicting data is concerned. The main goal for synchronization is: *GT must not read data that can be subject to compensating transactions*. To reach that goal the following rules are introduced:

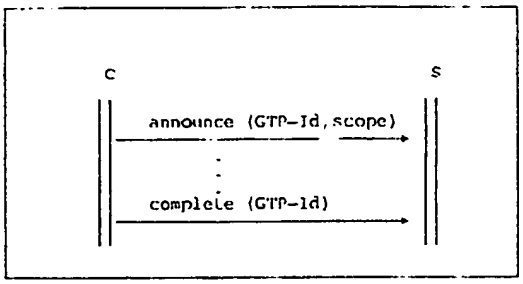


Fig. 8. *Announcement and Completion of GTP*: A Client (C) announces a subtransaction of a GTP by identifying the GTP with an unique GTP-ID. In addition, the scope of data to be accessed is announced at server S. The GTP is deactivated at S when a *complete* has been received, signaling that all subtransactions of that particular GTP are successfully finished.

1. Intended subtransactions of GTs and GTPs are *announced* to nodes to be accessed (*Fig. 8*).
2. Conflicting subtransactions are detected by scope descriptions of the *announce* message.
3. Data written by active GTPs can be read only by other active GTPs.
4. Data written by active GTPs can not be read by active GTs (and vice versa).
5. GTP waits for completion of conflicting GTs (and vice versa).
6. Completion of GTPs is announced to nodes concerned.

Failures of GTPs or unsuccessful subtransactions are dealt with by calling alternative or compensating subtransactions. The latter case requires deactivating subtransactions at each node concerned.

Since we know that 2PL/2PC provides for global serializability, all applications running within that synchronization/recovery technique are correct.

On the other hand all GTPs are build out of subtransactions that are locally order independent.

The combination of both avoids the reading of dirty data by subtransactions of GT. Thus all executions of GTs are correct.

Note that correctness means that local databases are always in a consistent state. This is guaranteed by the fact that all subtransactions of

8?61 Future Generations Computer Systems
7(1992)April, Nos.2/3, Amsterdam, NL

GT and GTP are handled by the local concurrency control mechanism. Moreover from a global viewpoint order independence guarantees that all compensating subtransactions will compensate the effects of their intended subtransaction counterparts.

From a global viewpoint the user has to consider whether the subtransactions exported by server nodes are sufficient for his task then executed by a Global Transaction Procedure. Otherwise he has to select the Global Transaction for his purposes.

6. Conclusion

Since both the transaction models of type GT and GTP introduced in this paper, show some drawbacks, we have proposed their combination into one environment.

The potential user therefore can chose (within the limits shown) between two models of transactions. Furthermore, the server administrator is able to chose whether he wants to open his database for unrestricted use by remote users, or restrict the application to prepared subtransactions as part of Global Transaction Procedures with reduced isolation properties. The latter will provide him with a higher degree of autonomy.

Acknowledgements

This work was done in a joint research project of the European Networking Center of IBM in Heidelberg and the University of Frankfurt.

References

[1] H. Garcia-Molina and B. Kogan, Node autonomy in distributed systems, in: *Proc. Internat. Symp. on Databases in Parallel and Distributed Systems*, Austin TX (1988).

[2] W. Johannsen and W. Lamersdorf, An open systems architecture for transaction supported distributed database applications, in: *Proc. 2nd Workshop on Future Trends of Distributed Computing Systems in the 1990's*, Cairo (1990).

[3] K.P. Eswaran, J.N. Gray, P.A. Lorie and I.L. Traiger, The notions of of consistency and predicate locks in a database system, *Commun. ACM* 19 (11) (1976) 624–633.

[4] J. Gray, Notes on data base operating systems. IBM Research Report RJ2188, 1978.

[5] A.P. Shet and M.T. Liu, Integrating locking and optimistic concurrency control in distributed database systems, in: *Proc. 6th Conf. on Distributed Computing Systems*, Cambridge, MA (1986).

[6] W. Du and A.K. Elmagarmid, A paradigm for concurrency control heterogeneous distributed database systems, Technical Report, Purdue University, Computer Science Department, CSD-TR-894, 1989.

[7] W. Du and A.K. Elmagarmid, Quasi serializability: a correctness criterion, for global concurrency control in Interbase, in: *Proc. VLDB Annual Conf.*, Amsterdam (1989) 347–355.

[8] H. Garcia-Molina and K. Salem, Sagas, in: *Proc. ACM SIGMOD Annual Conf.* San Francisco, CA (1987) 249–259.

[9] F. Eliassen and J. Veijalainen, An S-transaction definition language and execution mechanism, Working Papers of Gesellschaft fuer Mathematik und Datenverarbeitung, Bonn St. Augustin, 1987.

[10] J. Klein and A. Reuter, Migrating transactions, in: *Proc. Workshop on the Future Trends of Distributed Computing Systems in the 1990s*. Hong Kong (1988).

[11] H. Garcia-Molina, Using semantic knowledge for transaction processing in a distributed database, *ACM Trans. Database Syst.* 8 (2) (1983) 186–213.

[12] International Organization for Standardization, Basic Reference Model IPS-OSI, International Standard 7498, 1984.

[13] International Organization for Standardization, Remote Database Access, Part 1: Service and Protocol IPS-OSI, Draft Proposal 9579-1 (ISO/IEC JTC1/C21 WG3 N845), 1990.

[14] International Organization for Standardization, Distributed Transaction Processing Information Processing Systems -- Open Systems Interconnection, IPS-OSI, Draft International Standard 10026-1, 10026-2, 10026-3, 1989.

[15] P.A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems* Addison-Wesley Series in Computer Science, Reading, MA, 1987).

[16] GMD-FOKUS, INRIA, SWIFT Final Report MAP Project 761B 'Multidatabase Services on ISO/OSI Networks for Transnational Accounting'. SWIFT (eds.), May 1989.

[17] R. Alonso, H. Garcia-Molina and K. Salem, Concurrency Control and Recovery for Global Procedures in Federated Database Systems, *Data Engrg.* 10 (3) (1987) 5–11.

[18] J.W. Schmidt, F. Matthes and H. Eckhardt, DBPL-Report, DBPL-Memo 112-88, University of Frankfurt, 1988.

[19] W. Johannsen, Transaktionen in foederativ verteilten Datenbanken, PhD Thesis, University of Frankfurt, 1990.

Wolfgang Johannsen received the MS. in computer science from University of Hamburg, Germany in 1983.

He worked as a research assistant from 1984 to 1987 at University of Hamburg and from 1987 to 1989 at University of Frankfurt, Germany where he received the PhD. in computer science.

In 1989 he joined the IBM European Networking Center in Heidelberg. His major areas of research is distributed transaction processing. In addition he has been concerned with the standardization of the Transaction Processing protocol as a member of DIN and ISO. Since 1991 he is working in the EUREKA PROMETHEUS project in the area of travel information systems.